```
/*****************************************************************************
 *                                                                           *
 *                                                                           *
 *          ALL RIGHTS RESERVED                                              *
 *                                                                           *
 *****************************************************************************
 *                                                                           *
 *       PROGRAM NAME:  fontcheck                                            *
 *       RELEASE: 6.00                                                       *
 *       DATE:    Dec 11, 1998                                               *
 *                                                                           *
 *****************************************************************************
 *                                                                           *
 *       AUTHOR:  Mike Martyn                                                *
 *       DIVISION: SDS - Engineering Productivity Group                      *
 *       PROJECT MANAGER: Cliff Lindroth                                     *
 *                                                                           *
 *****************************************************************************
 *                                                                           *
 * Program Description:                                                      *
 *                                                                           *
 * REV 1: 9-7-93                                                             *
 *  This program takes as input an .mi file. It scans for problem characters *
 *  that will not translate in the mi2dxf translator. It counts the number   *
 *  of offenders and changes their color in the resultant modified .mi file. *
 *  The count of offenders is output to the user and on the next load of the *
 *  .mi file into me30, the offenders will show up in a special color.       *
 * REV 2: 10-12-93                                                           *
 *  The offenders are now tagged in the .mi file with a ptr to an info       *
 *  element that specifies a layer. So when the new_mifile is loaded into    *
 *  me30 one can toggle the badtext layer to see all of the elements that    *
 *  need to be converted to vectors before mi2dxf translation.               *
 *  Bad prefixes and postfixes to dimensions are found now as well.          *
 * REV 3: 8-3-94                                                             *
 *  The change to the mi file format with me30 6.0 required the change from  *
 *  "four" to "five" lines being used after the TEX sequence number.         *
 * REV 4: 8-12-94                                                            *
 *  Added the "make_newline" function to "swap out" the diameter, degree,    *
 *  and plus/minus (^N) characters before tagging "badtext" to the badtext   *
 *  layer. So as to convert fewer items to vectors and reduce the size of the *
 *  resultant translated file.                                               *
 * REV 5: 12-11-95                                                           *
 *  Added the screening for BALLOON elements in the .mi file. Any text,      *
 *  whether its within a dimension or not; has symbol fonts or function key  *
 *  symbols or not, gets tagged for vectorization.                           *
 * REV 6: 12-11-98 (wow, what a coincidence, December 11th)                  *
 *  Added the screening for any TEXT that contains a control character _     *
 *  for the MI files generated in SolidDesigner's Anotation Module           *
 * REV 7: 02-26-99                                                           *
 *  Rewrite of the program. Added new functionality.                         *
 *   -new usage message to handle 3 options and 2 possible commandlines      *
 *   -original code becomes one of the options, with newly added flexibility *
 *    @tag    -tag vectorization elements with info given                    *
 *    @find   -return text of the TEX element(s) that point to the given info *
 *    @change -change text of the TEX element(s) that point to the given info *
 *             with the given new text.                                      *
 * REV 8: 01-17-02                                                           *
 *  Changed all arrays to have SIZE linelength to accomodate the multi line  *
 *  size of infos in the MI file. Currently set at 500.                      *
 *  This fixed a segmentation violation core dump that was happening with    *
 *  inqmi.
```

```
 *                                                                          *
 ****************************************************************************/

#include <stdio.h>

#define SIZE                            500
#define MATCH                           != NULL
#define NO_MATCH                        == NULL
#define my_itoa(string_p, x)            sprintf(string_p, "%d\n", x)

int make_element(char array[SIZE][SIZE], int *, char[SIZE], char[SIZE], FILE *,
char array2[SIZE][SIZE], int);
int tag_element(char array[SIZE][SIZE], int *, char[SIZE], int);
void make_newline(char *line, char *newline);
void usage(char[15], char[SIZE]);

main(argc, argv)
int             argc;
char           *argv[];
{

char option[10], filename[280], infotext[80], new_infotext[80], uname[80],
info_textfile[80];

/*============================== usage ==============================*/
  if (argc <= 2) {
    usage(argv[0],"");
  }else {
    if (strstr(argv[3], "tag") MATCH) {
      if (argc != 5) usage(argv[0],"tag");
    }else if (strstr(argv[3], "find") MATCH) {
      if (argc != 5) usage(argv[0],"find");
    }else if (strcmp(argv[3], "changem") == NULL) {
      if (argc != 5) {
        usage(argv[0],"changem");
      }
    }else if (strcmp(argv[3], "change") == NULL) {
      if (argc != 6) {
        usage(argv[0],"change");
      }
    }else {
      usage(argv[0],"");
    }
  }

/*============================== MAIN ==============================*/
  strcpy(filename, argv[1]);
  sprintf(uname, "%s", argv[2]);
  strcpy(option, argv[3]);
  sprintf(infotext, "%s", argv[4]);
  sprintf(new_infotext, "%s\n", argv[5]);
  sprintf(info_textfile, "%s", argv[4]);

printf("%s %s %s %s %s\n", filename, uname, option, info_textfile,
new_infotext);

  if (strstr(option, "tag") MATCH) {
    printf ("%s\n", "going into function tag");
    tag_option(filename, infotext, uname);
  }
  else if (strstr(option, "find") MATCH) {
```

```
    printf ("%s\n", "going into function find");
    find_option(filename, infotext);
  }
  else if (strcmp(option, "change") == NULL) {
    printf ("%s\n", "going into function change_option");
    change_option(filename, infotext, new_infotext);
  }
  else if (strcmp(option, "changem") == NULL) {
    printf ("%s\n", "going into function change_option2");
    change_option2(filename, info_textfile);
  }
}

/*================== MAIN_TAG_option ==============================*/
int tag_option (filename, infotext, uname)
  char filename[80], infotext[80], uname[80];

    {
    FILE  *mifile, *new_mifile, *tmp_new_mifile;
    int i=0, el_count=0, bal_count=0;
    int one=1, four=4, five=5, six=6, *array_count, n_args;
    char line[SIZE], newline[SIZE], element[SIZE][SIZE], bals[SIZE][SIZE];
    char array[SIZE][SIZE], info_tag[10], unamefile[80];

    mifile = fopen(filename, "r");
    sprintf(unamefile, "%s%s", "/tmp/one_", uname);
    tmp_new_mifile = fopen(unamefile, "w");

    while(fgets(line, sizeof(line), mifile) != NULL) {
      if(((strstr(line, "\016\060\017") MATCH)) ||
         ((strstr(line, "\016\061\017") MATCH)) ||
         ((strstr(line, "\016\062\017") MATCH))) {
        make_newline(line, newline);
        fputs(newline, tmp_new_mifile);
      }
      else
        fputs(line, tmp_new_mifile);
      if(strstr(line,"BAL\n") MATCH) {
        fgets(line, sizeof(line), mifile);
        fputs(line, tmp_new_mifile);
        strcpy(bals[bal_count], line);
        ++bal_count;
      }
    }
    --bal_count;
    fclose (mifile);
    fclose (tmp_new_mifile);

    strcpy(info_tag, "");
    mifile = fopen(unamefile, "r");
    sprintf(unamefile, "%s%s", "/tmp/tagged_mi_", uname);
    new_mifile = fopen(unamefile, "w");

    while(fgets(line, sizeof(line), mifile) != NULL) {
      fputs(line, new_mifile);
      el_count=0;

    /*   if ((strcmp(line, "BSPL\n") == NULL) ||
           (strcmp(line, "SPL\n") == NULL)) {
        make_element(element,&el_count,info_tag,infotext,mifile,bals,bal_count);
        el_count = tag_element(element, &el_count, info_tag, five);
```

```
        for(i=0;i<el_count;i++)
          fputs(element[i], new_mifile);
        fputs("|~\n", new_mifile);
      }
      else */

      if (strcmp(line, "LED\n") == NULL) {
        make_element(element,&el_count,info_tag,infotext,mifile,bals,bal_count);
        el_count = tag_element(element, &el_count, info_tag, five);
        for(i=0;i<el_count;i++)
          fputs(element[i], new_mifile);
        fputs("|~\n", new_mifile);
      }

    if((strcmp(line, "TEX\n") == NULL) ||
              (strcmp(line, "ASSP\n") == NULL)) {
        if ((strcmp(line, "TEX\n") == NULL) && (strcmp(info_tag, "") == NULL)) {
          printf ("Info supplied does not exist in this MI file.\n");
          exit(1);
        }

    if(make_element(element,&el_count,info_tag,infotext,mifile,bals,bal_count))
          el_count = tag_element(element, &el_count, info_tag, five);
        for(i=0;i<el_count;i++)
          fputs(element[i], new_mifile);
        fputs("|~\n", new_mifile);
      }

      else if((strcmp(line, "DSGS\n") == NULL) ||
              (strcmp(line, "DRAD\n") == NULL) ||
              (strcmp(line, "DDIA\n") == NULL) ||
              (strcmp(line, "DANG\n") == NULL) ||
              (strcmp(line, "DARC\n") == NULL) ||
              (strcmp(line, "DSGL\n") == NULL) ||
              (strcmp(line, "DCHN\n") == NULL) ||
              (strcmp(line, "DDAS\n") == NULL) ||
              (strcmp(line, "DDAL\n") == NULL) ||
              (strcmp(line, "DCOR\n") == NULL) ||
              (strcmp(line, "DDLS\n") == NULL) ||
              (strcmp(line, "DCHMF\n") == NULL))  {
     if(make_element(element,&el_count,info_tag,infotext,mifile,bals,bal_count))
          el_count = tag_element(element, &el_count, info_tag, one);
        for(i=0;i<el_count;i++)
          fputs(element[i], new_mifile);
        fputs("|~\n", new_mifile);
      }else
        continue;
  }
  fclose (mifile);
  fclose (new_mifile);
return(0);
}

/*================== MAIN_FIND_option ================================*/
int find_option (filename, infotext)
  char filename[80], infotext[80];
{
  FILE  *mifile;
  int e=0, i=0, m=0, n=0, count=0, bal_count=0, found=0, info_count=0, text_loc;
  int one=1, four=4, five=5, six=6, *array_count, ar_len=0, n_args, d=0;
  int info_index=0, imatch_count=1, size_str=0;
```

```
  char line[SIZE], bominfo[SIZE], earray[SIZE][SIZE], bals[SIZE][SIZE];
  char info_array[SIZE][SIZE], array[SIZE][SIZE], info_tag[10], el_type[10];
  char info_matches[SIZE][SIZE], info_textmatches[SIZE][SIZE];

  char text[260];

  mifile = fopen(filename, "r");

/* START READING THE MI FILE */
  while(fgets(line, sizeof(line), mifile) != NULL) {
    count=0;

    strcpy(el_type, line);
    if((strcmp(el_type, "ASSP\n") == NULL) ||
       (strcmp(el_type, "TEX\n") == NULL)) {
/* element type is ASSP or TEX */
      if ((strcmp(el_type, "TEX\n") == NULL) && (imatch_count == 1)) {
/*
found a TEX element and imatch_count is still empty: didn't find the info tag
in the infos ASSPs so exit with message "info not found"
*/
        printf ("Info supplied does not exist in this MI file.\n");
        exit(1);
      }

/* ENTERING A TEX or ASSP ELEMENT */
      while(strstr(fgets(line, sizeof(line), mifile), "|~\n") NO_MATCH) {
/* START building ELEMENT array */
        strcpy(earray[count], line);
        if(count == 5){
/* capture the info array */
/*get the value of earray[count] this is the number of infos in the element*/
          info_count = atoi(earray[count]);
          for (info_index=0;info_index<info_count;info_index++,count++){
            fgets(line, sizeof(line), mifile);
/* CONTINUE building ELEMENT array */
            strcpy(earray[count], line);
/* START building INFO array */
            strcpy(info_array[info_index], line);
          }
/* element's info_array is now captured and some of the element array */
        }else{
          count++;
        }

/*compare each line with the search text (infotext) for contained in*/
        if(strstr(line, infotext) MATCH) {  /* MATCH means "contained in" */
/* if you match the infostring you are in the info ASSP you are looking for */
/*printf("line is :%s->infotext is: %s\n",line, infotext);*/

/*sprintf(info_tag, "%d", earray[0]); */
/*** earray[0] needs to go into the info_matches array (not info_tag string) */
/*        strcpy(info_tag, earray[0]);  */

/*printf("imatch_count is: %d\n",imatch_count);*/
/*printf("earray sub0 is: %s\n",earray[0]);*/
          strcpy(info_matches[imatch_count], earray[0]);
          strcpy(info_textmatches[imatch_count], earray[2]); /*chomp*/
/*printf("info_matches sub%d is:
%s\n",imatch_count,info_matches[imatch_count]);*/
/*this captures the info element number*/
```

```
/* of the infotext being searched for */
/* printf("the info element number is *%s*",info_matches[imatch_count]);*/
          imatch_count++;
        }
      }
/*
      printf("*info_matches sub0 is: %s",info_matches[0]);
      printf("*info_matches sub1 is: %s",info_matches[1]);
      printf("*info_matches sub2 is: %s",info_matches[2]);
*/


/*=================NEEDS TESTING FROM HERE==============================*/
/* FINISHED AN ELEMENT */

/* the TEX element has now been read into the earray */
/* printf("%s: %d","length of element earray is",count); */

      if (strcmp(el_type, "TEX\n") == NULL) {
/* again check if element is TEX */
        for(m=0;m<info_count;m++) {
/* printf("info_matches sub%d is: *%s*",m,info_matches[m]); */
/*printf("the info element number is *%s*",info_matches[imatch_count]);*/

/* for each index in the info_array check the info number to see if it
 matches any of the info_matches array's info_tag numbers */
          for(n=0;n<imatch_count;n++) {
/*   printf("info_matches sub%d is *%s*",n,info_matches[n]);*/
            if (strcmp(info_array[m], info_matches[n]) == NULL) {
/* exactly matched the info_tag to an info in the info_array of this element */
              if (strstr(earray[count -2], "_") == NULL) {
                d=0;
                while(info_textmatches[n][d] != '\n'){
                  bominfo[d] = info_textmatches[n][d];
                  d++;
                }
                bominfo[d] = NULL;
/*
                size_str = strlen(info_textmatches[n]);
                strcpy(bominfo,"");
                for(d=0;d<size_str-1;d++) {
                  bominfo[d] = info_textmatches[n][d];
                }
                bominfo[d] = NULL;
*/
/* print out the text (2 lines up from the bottom of the array) */
                for(e=27+info_count;e<=count-2;e++) {
                  /*printf("%s:%s", bominfo,earray[count-2]);*/
                  printf("%s:%s", bominfo,earray[e]);
                }
                strcpy(bominfo,"");
              }
              break;
            }
          }
        }
      }
    }
  }
/*
  for (i=0;i<info_count;i++) {
```

```
    printf ("%s",info_array[i]);
  }
  for (i=0;i<imatch_count;i++) {
    printf ("%s",info_matches[i]);
  }
*/
  fclose (mifile);
  return(0);
}

/*================== MAIN_CHANGE ============================*/
int change_option (filename, infotext, new_infotext)
  char filename[80], infotext[80], new_infotext[80];
{
  FILE  *mifile, *new_mifile;
  int i=0, e=0, count=0, bal_count=0, found=0, info_count=0, info_index=0,
text_loc;
  int one=1, four=4, five=5, six=6, *array_count, ar_len=0, n_args;
  int textLineCount=0;
  char line[SIZE], newline[SIZE], element[SIZE][SIZE], bals[SIZE][SIZE];
  char info_array[SIZE][SIZE], array[SIZE][SIZE], info_tag[10], el_type[10];

  char text[260];

  printf ("infotext is: %s\n", infotext);
  printf ("new_infotext is: %s", new_infotext);

  strcpy(info_tag, "");
  mifile = fopen(filename, "r");
  new_mifile = fopen("/tmp/changedMI", "w");

/* START READING THE MI FILE */
/*============================*/
  while(fgets(line, sizeof(line), mifile) != NULL) {

    count=0;
    info_count=0;
    strcpy(el_type, line);

/* ELEMENT TYPE IS ASSP or TEX */
/*=============================*/
    if((strcmp(el_type, "ASSP\n") == NULL) ||
       (strcmp(el_type, "TEX\n") == NULL)) {
      if ((strcmp(el_type, "TEX\n") == NULL) && (strcmp(info_tag, "") == NULL))
      {
        printf ("Info supplied does not exist in this MI file.\n");
        exit(1);
      }

/* BUILD THE ELEMENT ARRAY */
/*=========================*/
      while(strstr(fgets(line, sizeof(line), mifile), "|~\n") NO_MATCH) {
        strcpy(array[count], line);
        if (strcmp(el_type, "TEX\n") == NULL) {
/* IF TEX ELEMENT, BUILD THE INFO_ARRAY */
/*======================================*/
          if(count == 5){
            info_count = atoi(array[count]);
            ++count;
            for (info_index=0;info_index<info_count;info_index++,count++){
              fgets(line, sizeof(line), mifile);
```

```
              strcpy(array[count], line);
              strcpy(info_array[info_index], line);
            }
          }else{
            ++count;
          }
        }
/* IF ASSP ELEMENT, CHECK FOR infotext, capture ASSP ELEMENT NUMBER (info_tag)
*/
/*=============================================================================*
/
        if (strcmp(el_type, "ASSP\n") == NULL) {
          if(strstr(line, infotext) MATCH) {
            strcpy(info_tag, array[0]);
          }
          ++count;
        }
      }
    }else{
      fputs(line, new_mifile);
    }

/*COMPARE INFO_TAG TO INFO_ARRAY TAGS */
/*====================================*/
    if (strcmp(el_type, "TEX\n") == NULL) {
      for(i=0;i<info_count;i++) {
/*IF MATCH, MAKE THE TEXT CHANGE IN THE ELEMENT*/
/*============================================*/
        if (strcmp(info_array[i], info_tag) == NULL) {
          /*
          printf("%s%s", info_array[i], info_tag);
          printf ("strcmp info_array[%d]=%s info_tag=%s", i, info_array[i],
info_tag);
          printf("oldText was: %s", array[count - 2]);
          */

          /* this is new 2-04-02, change the text */
          /* if the number of text lines is more than one, */
          /* this loop is blanking out the multiline text */

                e=27+info_count;
                textLineCount = atoi(array[e]);
                if(textLineCount > 1) {
                  for(e=textLineCount+1;e<=count-2;e++) {
                    if(strcmp(array[e], "0\n") == NULL) {
                    }else{
                      strcpy(array[e], new_infotext);
                    }
                  }
                }else{
                   strcpy(array[count - 2], new_infotext);
                }

          /* the following line was in use instead of the 1-30-02 block above */
          /* strcpy(array[count - 2], new_infotext); */
          break;
        }
      }
/*WRITE OUT THE TEX ELEMENT TO THE NEW MI FILE*/
/*============================================*/
      fputs(el_type, new_mifile);
```

```
      for(i=0;i<count;i++) {
        fputs(array[i], new_mifile);
      }
      fputs("|~\n", new_mifile);
/*WRITE OUT THE ASSP ELEMENT TO THE NEW MI FILE*/
/*=============================================*/
    } else if (strcmp(el_type, "ASSP\n") == NULL) {
        fputs(el_type, new_mifile);
        for(i=0;i<count;i++) {
          fputs(array[i], new_mifile);
        }
        fputs("|~\n", new_mifile);
    }
  }
  fclose (mifile);
  fclose (new_mifile);
return(0);
}

/*================== MAIN_CHANGEM ============================*/
int change_option2 (filename, info_textfile)
  char filename[80], info_textfile[80];
{
  FILE  *mifile, *new_mifile, *infotextfile;
  int i=0, e=0, count=0, bal_count=0, found=0, info_count=0, info_index=0,
text_loc;
  int k=0, j=0, m=0, one=1, four=4, five=5, six=6, *array_count, ar_len=0,
n_args;
  int textLineCount=0, orgstringLength=0, n=0;
  char line[SIZE], newline[SIZE], element[SIZE][SIZE], bals[SIZE][SIZE];
  char line1[80], str1[SIZE], str2[SIZE];
  char info_array[SIZE][SIZE], array[SIZE][SIZE], info_tag[10], el_type[10];
  char infotext[80], new_infotext[80];
  char changem_array[100][80];

  char text[260];

  infotextfile = fopen(info_textfile, "r");
  strcpy(info_tag, "");

/*WORKING: HERE IS WHERE I SHOULD LOOP OVER THIS WHOLE THING
  FOREACH LINE IN THE INFOTEXTFILE
  EITHER THAT OR GO INTO THIS WHILE LOOP WHICH ITERATES OVER
  THE MIFILE, WITH AN ARRAY OF CHANGES?? YES ARRAY OF CHANNGES.
  ON EACH INSPECTION CHECK AGAINST THE COMPLETE ARRAY INSTEAD OF
  JUST THE ONE STRING.
*/
/* FOR EACH LINE IN THE INFO_TEXTFILE */
/* READ THE INFO_TEXTFILE INTO AN ARRAY*/
/* LOOP THROUGH THE MI FILE ONCE AND INSPECT THE ARRAY
   AT EACH COMPARISON POINT. */
/*====================================*/
  while(fgets(line1, sizeof(line1), infotextfile) != NULL) {
    /*changem_array[m] = line1; */
    /* how do I put a line into an array in C?? */
    j=0;
    found = 0;
    strcpy(infotext, "");
    strcpy(new_infotext, "");
    orgstringLength = strlen(line1);
    for (k=0;k<orgstringLength;k++) {
```

```
      if (line1[k] == ':') {
        found=1;
        k++;
      }
      if (found) {
        infotext[k-1] = '\0';
        new_infotext[j] = line1[k];
        j++;
      } else {
        infotext[k] = line1[k];
      }
    }
    new_infotext[j-1] = '\n';
    new_infotext[j] = '\0';

    printf ("infotext is: %s\n", infotext);
    printf ("new_infotext is: %s", new_infotext);
    m++;
  }
/*
  for (i=0;i<=m-1;i++) {
    printf("%s", changem_array[i]);
  }
*/
  exit(1);

  strcpy(info_tag, "");
  mifile = fopen(filename, "r");
  new_mifile = fopen("/tmp/changedMI", "w");

/* START READING THE MI FILE */
/*===========================*/
  while(fgets(line, sizeof(line), mifile) != NULL) {

    count=0;
    info_count=0;
    strcpy(el_type, line);

/* ELEMENT TYPE IS ASSP or TEX */
/*=============================*/
    if((strcmp(el_type, "ASSP\n") == NULL) ||
       (strcmp(el_type, "TEX\n") == NULL)) {
      if ((strcmp(el_type, "TEX\n") == NULL) && (strcmp(info_tag, "") == NULL))
      {
        printf ("Info supplied does not exist in this MI file.\n");
        exit(1);
      }

/* BUILD EACH ELEMENT's ARRAY */
/*============================*/
      while(strstr(fgets(line, sizeof(line), mifile), "|~\n") NO_MATCH) {
        strcpy(array[count], line);
        if (strcmp(el_type, "TEX\n") == NULL) {

/* IF TEX ELEMENT, BUILD THE ELEMENT's INFO_ARRAY */
/*================================================*/
          if(count == 5){
            info_count = atoi(array[count]);
            ++count;
            for (info_index=0;info_index<info_count;info_index++,count++){
              fgets(line, sizeof(line), mifile);
```

```
            strcpy(array[count], line);
            strcpy(info_array[info_index], line);
          }
        }else{
          ++count;
        }
      }

/* IF ASSP ELEMENT, CHECK FOR infotext, capture ASSP ELEMENT NUMBER (info_tag)
*/
/*=============================================================================*
/
      if (strcmp(el_type, "ASSP\n") == NULL) {
        if(strstr(line, infotext) MATCH) {
          strcpy(info_tag, array[0]);
        }
        ++count;
      }
    }
  }else{
    fputs(line, new_mifile);
  }

/*IF ELEMENT IS TEX, COMPARE INFO_TAG TO INFO_ARRAY TAGS */
/*=====================================*/
  if (strcmp(el_type, "TEX\n") == NULL) {
    for(i=0;i<info_count;i++) {

/*IF MATCH, MAKE THE TEX VALUE CHANGE IN THE ELEMENT*/
/*================================================*/

      if (strcmp(info_array[i], info_tag) == NULL) {
        printf("%s%s", info_array[i], info_tag);

        printf ("strcmp info_array[%d]=%s info_tag=%s", i, info_array[i],
info_tag);
        printf("oldText was: %s", array[count - 2]);

        /* this is new 2-04-02, change the text */
        /* if the number of text lines is more than one, */
        /* this loop is blanking out the multiline text */
        /*
            e=27+info_count;
            textLineCount = atoi(array[e]);
            if(textLineCount > 1) {
              for(e=textLineCount+1;e<=count-2;e++) {
                if(strcmp(array[e], "0\n") == NULL) {
                }else{
                  strcpy(array[e], new_infotext);
                }
              }
            }else{
               strcpy(array[count - 2], new_infotext);
            }
         */
        /* the following line was in use instead of the 1-30-02 block above */
         strcpy(array[count - 2], new_infotext);
        break;
      }
    }
 /*WRITE OUT THE TEX ELEMENT TO THE NEW MI FILE*/
```

```
/*=============================================*/
      fputs(el_type, new_mifile);
      for(i=0;i<count;i++) {
        fputs(array[i], new_mifile);
      }
      fputs("|~\n", new_mifile);
/*WRITE OUT THE ASSP ELEMENT TO THE NEW MI FILE*/
/*=============================================*/
    } else if (strcmp(el_type, "ASSP\n") == NULL) {
        fputs(el_type, new_mifile);
        for(i=0;i<count;i++) {
          fputs(array[i], new_mifile);
        }
        fputs("|~\n", new_mifile);
    }
  }
  fclose (mifile);
  fclose (new_mifile);
/*
  system ("cp /tmp/changedMI /tmp/changedMI2");
  filename = "/tmp/changedMI2";
 } */
  fclose (infotextfile);
return(0);
}

/* ================ MAKE ELEMENT FUNCTION ========================
 * MAKE NEW ELEMENTS. FIND IF ELEMENT IS BAD, IF SO ADD PTR TO
 * BADTEXT ASSP, INC PTR COUNT, PUT THE NEW ELEMENT INTO THE NEW_MIFILE.
 * TAKES IN: ELEMENT ARRAY, ARRAY INDEX PTR, NUMBER OF BADTEXT ASSP, MIFILE,
  mifile = new_mifile;
 * BAL ARRAY, INT_COUNTER.
 */

int make_element(array,array_count,info_tag,infotext,mifile,bals,bal_count)
char array[SIZE][SIZE];
int *array_count;
char info_tag[10], infotext[80];
FILE *mifile;
char bals[SIZE][SIZE];
int  bal_count;
{
   int badtext=0, i;
   char line[SIZE];

      while(strstr(fgets(line, sizeof(line), mifile), "|~\n") NO_MATCH) {
            strcpy(array[*array_count], line);
            ++*array_count;

       /*  if(strstr(line, "badtext")  MATCH) */ /*this works*/
            if(strstr(line, infotext)  MATCH)       /*this works*/
         strcpy(info_tag, array[0]);

 /*Check to see if this element contains any of the following pieces of
   text and if so indicate it needs to be tagged */

            if((strstr(line, "_y14") MATCH) ||
          (strstr(line, "_Y14") MATCH) ||
          (strstr(line, "_")   MATCH) ||
          (strstr(line, "_d_") MATCH) ||
          (strstr(line, "_e_") MATCH) ||
```

```
          (strstr(line, "_f_") MATCH) ||
          (strstr(line, "_g_") MATCH) ||
          (strstr(line, "_h_") MATCH) ||
          (strstr(line, "_0_") MATCH) ||
          (strstr(line, "_1_") MATCH) ||
          (strstr(line, "_2_") MATCH) ||
          (strstr(line, "symb") MATCH))
             badtext=1;

/*Check to see if this element points to one of the BAL elements (recorded
  in the bal[] array) and if so indicate it needs to be tagged. */

      for(i=0;i<=bal_count;++i)
        if(strcmp(line,bals[i]) == NULL) badtext=1;
   }
return(badtext);
}

/* ================= TAG ELEMENT FUNCTION ========================
 * ADJUST THE ELEMENT BY INSERTING THE INFO TAG PTR TO BADTEXT ASSP AND
 * INCREMENT THE OTHER PTR #S IN THE ELEMENT.
 */

int tag_element(array, array_count, layer_ptr, num)
char array[SIZE][SIZE], layer_ptr[10];
int *array_count, num;
{
  int i, ptr_int;
  char ptr_char[80];

  ptr_int = atoi(array[num]) +1;
  my_itoa(ptr_char, ptr_int);

/*    MOVE ALL ARRAY VALUES FROM ARRAY[num+1] TILL EL_COUNT, UP ONE
 *    THEN COPY BADTEXT INFO PTR INTO ARRAY[num+1] AND ADD ONE TO THE EL_COUNT
 */

  strcpy(array[num], ptr_char);
  for(i=*array_count;i>=num;i--)
       strcpy(array[i + 1], array[i]);
  strcpy(array[num+1], layer_ptr);
  return(*array_count +1);
}

/* ================= SUBSTITUTE CHAR FUNCTION ======================*/
void make_newline(char *line, char *newline) {
  while(*line != '\0') {
    if(*line == '\016') {
      line++;
      switch (*line) {
        case '\060': *newline = '\362'; break;      /* 322'; */
        case '\061': *newline = '\363'; break;      /* 362'; */
        case '\062': *newline = '\376'; break;      /* 376'; */
        default:     *newline++ = '\016';
                     *newline++ = *line;
                     *newline = '\017';
      }
      newline++;
      line += 2;
    }
    else
```

```
      *newline++ = *line++;  /*increment the pointer position and */
  }        /*assign its content to the incrimented newline ptr's content. */
  *newline++ = '\0';         /*add a null to last char of newline */
}

/* ================ USAGE =======================================*/
void usage(char command[15], char option[80]) {

  if ((strstr(option, "tag") MATCH))
    printf("%s %s %s\n","\nUsage:",command,"<MI file> uniqID tag
<info_text>\n");
  else if ((strstr(option, "find") MATCH))
    printf("%s %s %s\n","\nUsage:",command,"<MI file> uniqID find
<info_text>\n");
  else if ((strstr(option, "changem") MATCH))
    printf("%s %s %s\n","\nUsage:",command,"<MI file> uniqID changem
<info_textfile>\n");
  else if ((strstr(option, "change") MATCH))
    printf("%s %s %s\n","\nUsage:",command,"<MI file> uniqID change <info_text>
\"new TEX value\"\n");
  else
    printf("%s %s %s\n","\nUsage:",command,"<MI file> uniqID tag|find|change
<info_text> [\"new TEX value\"]\n");
  exit(1);
}
```